This is the text I will be using. When this is final, I will be copying the notes into the slidedeck.

The Future of the Portal and uPortal 5

Jim Helwig at ESUP Days #23

Introduction

I am Jim Helwig, Chair of the uPortal Steering Committee. I also serve on the Apereo Board of Directors and have been involved with Apereo and formerly Jasig since 2003. I have been employed at the University of Wisconsin-Madison since 2001 and ever since have been part of the team behind MyUW, our campus enterprise portal. My current position is that of Lead Strategist for the team. My primary role is to look for collaboration opportunities by working with partners on the Madison campus, throughout the University of Wisconsin System, and beyond.

Emerging Themes

Before I focus in on what is new with uPortal, I'll start by sharing some general themes that seem to be emerging around portals. I'll then talk about some of the latest functionality in uPortal and the improved build architecture. I'll end the hour with a discussion on how we can further collaborate to go even further. There should be time at the end for questions.

Classic Portals

But first a quick review of how we got here. Portals started to emerge in the late 1990s first with web portals like Yahoo! and then commercial products like Epicentric as well as offerings from BEA, IBM, and others. Around that time a number of universities started development of uPortal and by the early 2000s it had been deployed by many campuses. Early on uPortal was distinguished by its powerful groups and permissions system and dynamic layouts which were a particularly good fit for the multiple user roles found on campuses and in 2007 uPortal was recognized with the EDUCAUSE Catalyst Award as "the leading open source enterprise portal framework built by and for the higher education community."

Ten years ago was perhaps the most energetic time in the portal marketplace. There were at least a dozen different products available but there were some common characteristics. Most had a layout consisting of multiple portlets on tabs. Most had built-in rudimentary content management systems. Most came bundled with simple collaboration tools (e.g., polls, discussion boards). Most were locally hosted. But the world has changed over the last ten years. The expectations of our users is higher and but university resources continue to be squeezed. There are four trends in particular I would like to highlight.

Importance of Mobile

First is the importance of mobile. Ten years ago less that 5% of web usage was through mobile devices yet today some estimates put it at over 50%. At UW-Madison, mobile and tablet access is closing in on 20% for MyUW. Increasingly our community expects to be able to access important web resources from their smartphone or tablet. They are also expecting it to be a user friendly experience. The next generation portals are making this easier than ever. The modern user interface is implemented in a mobile-first, responsive manner. This is important for two reasons. First, the user experience is consistent across all devices; no longer is the mobile version completely different from the desktop version. Secondly, because the responsive implementation uses the same code base, it is developed with the same priority; no longer is the mobile version an afterthought or lagging.

Dynamic and Fast User Interfaces

The second trend is dynamic and fast user interfaces. In the classic portal, the page load was not complete until every portlet finished rendering. Every interaction with a portlet required a round trip. Today users expect a much faster response for both the initial page load and subsequent interactions. Luckily we have a growing number of open source Javascript libraries to use and more powerful devices to execute the client side code. By using AJAX and a modern REST architecture, user interactions no longer require a heavy page load.

Adoption of Common Paradigms

The third trend is the portal aligning with other common paradigms. Previously, the success of the classic portal led to many tabs with many portlets on each tab. This was a very busy interface that in the end was

often confusing to users. We regularly found out about students that were unaware of some of the more valuable content in MyUW. They were not hanging out in the portal or browsing content in the portal; they came to the portal with a specific goal and wanted to easily take action. The next generation portal supports this. In part this is accomplished by adopting the paradigms of other web resources such as effective search, apps and widgets, a customizable home page, and notifications.

Do More With Less

The final trend comes from the growing need to “do more with less.” While the number of staff resources dedicated to working on campus portals decreases, there is an ever increasing need to maintain existing content and add more. As campuses deploy new systems, there is an expectation that they be included in the campus portal. More users are gaining portal access. There is pressure to have the portal be support legacy portlets, be easy to administer, and even be offered as SaaS in the cloud.

New Functionality

I’ll spend the next few minutes in an overview of some of the newer functionality we have in uPortal that addresses these trends.

AngularJS-Portal

User research UW-Madison, Indiana University, and others indicates users want clean and simple interfaces. This was our inspiration around developing a new front end for uPortal called angularjs-portal. It is already adoptable with current uPortal as UW-Madison and Sinclair Community College are doing. It is currently going through Apereo incubation which we hope will be complete by the release of uPortal 5. The user interface went through many changes early on based on user feedback.

One of the key decisions we made was the adoption of Material Design. Google has put a lot of resources into the creation of these design patterns and they are increasingly being adopted. By adopting them, there is less design work for us to do.

So let’s see what this actually looks like. Similar to the icons and widgets you see on your phone or tablet, when you log into uPortal you see a homepage with widgets or links to apps. Because we know your identity and campus roles, we are able to present you with a default selection of widgets we believe will be the most useful.

The widgets themselves can be simply icon links or contain dynamic content. For example, I can quickly see whether or not my latest payroll statement is available, what the balance is on my Wiscard, or if there are any new scholarship opportunities.

If I need additional information than what is displayed in the widget, I can click on it to access the corresponding app. By clicking into the app, I indicate I'm engaging with that task or domain, with that particular aspect of my university experience. So we task-optimize. In this example I wanted to access payroll information and the app is relatively simple but apps can be more complex as needed for the task.

Here is an example of a slightly more complex app that allows the user to update their emergency contact information. That app becomes the whole experience and uses the whole browser window to give me the best possible experience over that particular task. This is a more modern experience than that of our legacy portal.

Here is an example of an even more complex app used by advisors. There are more elements to this user interface because the task of advising is more complex, yet this dashboard simplifies the advisor's task by pulling together information from a variety of systems.

If I don't see a widget relating to what I need, there is a prominent search bar.

This search aggregates results from the catalog of apps within the portal, the campus directory, and the campus search engine. If there is not yet a related app in the portal, chances are a useful link will show up in the campus web search. If I see an app that is useful I can either access it immediately or add it to my home page for quick access later.

Widgets added to my home page can be easily rearranged to put the most useful things first. I can switch out of expanded mode if I want to see more widgets without the dynamic content.

The user interface is also fully responsive so it scales down appropriately on a mobile device or even just in a smaller desktop browser window.

There is also a notification icon in the header. By clicking on that, I can see the list of unread notifications. Critical notifications are immediately displayed above the header for increased visibility. At UW-Madison we have developed a set of guidelines for notifications and are reserving this space for action-oriented, important notices.

Apps can do their own messaging either in the widget or in their full page view. This is often a more appropriate and contextualized place for the message, further reserving the global notification for more important notifications.

For special announcements, generally related to new content available within MyUW, a whimsical macot peeks out in the header. Clicking on it displays a brief announcement with a link for more information.

One of the notable differences of this new design is that there is relatively little on the default view and we got rid of tabs. The emphasis is on search and customization. The user isn't overwhelmed with an overload of content, most of which is not relevant to the task they are trying to accomplish. Another benefit of this design is that it scales beautifully. There can be hundreds of widgets, apps, or pieces of content but searching immediately narrows it down to those that are the most relevant.

Our system has scaled well as we expanded it to serve the other dozen campuses across the University of Wisconsin System. Each campus has its own skin and can contain content specifically targeted to that campus. A number of these campuses are starting to leverage the framework to develop additional content in their portal.

A number of schools have been interested in this new interface. Sinclair Community College is currently implementing this for their campus. They have used a hybrid approach and added some tab-like menuing to the user interface. We anticipate that easier adoption with uPortal 5 will result in more universities adopting it but the code is available now for those that want to start.

You can support both angularjs-portal and Respondr at the same time. In our case, we had them running in parallel for over a year until we finally turned off the classic Respondr view. This can give you a chance to beta test the new design with some users or allow for a graceful decommissioning of the classic view. Opt-in, opt-out, defaulting it to different users based on user identity or even the hostname by which they've reached the portal -- we did all of this on our gradual adoption so you can too.

To learn more about the motivation for this design, you can see the document from my presentation at Educause, https://goo.gl/PKsFQx "Beyond Portals: A Next Generation, Action-Oriented Service Delivery Platform for Rich Campus Applications."

AngularJS-portal is an alternative implementation of the portal landing page with some nice extras for notifications, announcements, and user settings (so far). It's opinionated about how these experiences work. If you want dynamic content on the homepage, you build a widget, not a Portlet.

But happily AngularJS-portal has no opinion about what happens once you click into a portlet, from a widget or from a search result. We're still running legacy JSR-286 portlets in MyUW and they work just fine maximized, rendered through the traditional rendering pipeline.

And so as I move on to talk about Soffits, a particular new JSR-286 portlet type in uPortal, bear in mind that so long as you're willing to use these only to deliver maximized, task-focused user experiences, Soffits are compatible with AngularJS-portal.

Soffits

Soffits are a new take on pluggable, custom content for the portal. They have similarities both with WSRP and Web Proxy. That is, Soffits are a way to deliver a Portlet in your portal that is proxying content and behavior provided by a remote web application. Whereas WSRP used SOAP and Web Proxy uses HTTP and HTML, Soffits uses ad-hoc JSON web services. This technology is simply much more flexible than strict JSR-286 portlets.

Soffits allow for using much newer tools to build services for your users to consume. The easiest way to do that is with Spring Boot (http://start.spring.io/). You can find some examples on Soffits with Spring Boot here: https://github.com/drewwills/soffit-samples Soffits work by using JSON Web Tokens (JWT https://jwt.io/) to pass uPortal information over to your Soffit. The JWT that is passed to your Soffit is also encrypted. This allows for more security for your end users.

One major advantage for Soffits is that they can be deployed outside of uPortal's tomcat container. If you have a really big portlet that sometimes takes down all of Tomcat, it would be a great idea to develop it as a Soffit. Now that you have a Soffit, you can load balance it so it never fully goes down! If your Soffits are still really small and you don't have a lot of infrastructure, they can be deployed in the same Tomcat container as uPortal. While deploying Soffits outside of uPortal's Tomcat is great, there is no standard way to manage that infrastructure. Docker would be a great way to handle this. If your institution is not ready for Docker, https://github.com/bpowell/brocker is another way of doing orchestration for your Soffits.

Here are some samples of Soffits in action. They can deliver the same classic portal look but have more flexibility on the backend.

Better Accessibility

Another feature now available in uPortal is better accessibility. In the fall of 2016 developers at Unicon performed an accessibility audit based on WCAG 2.0 Level AA. They identified a number of issues and worked on remediating them. They used the following guidelines:

- US - Section 504
- US - Section 508
- US - ADA
- FR - RGAA 3 -2016 (http://references.modernisation.gouv.fr/rgaa-accessibilite/)
- CA - AODA

You can read more about the test plan at https://wiki.jasig.org/x/CYBnB

The fixes are available now in uPortal 4.3 (rel-4-3-patches) and will be included in uPortal 5. Here is the specific issues identified

- https://issues.jasig.org/browse/UP-4735.
- https://issues.jasig.org/browse/EMAILPLT-190
- https://issues.jasig.org/browse/FBP-28
- https://issues.jasig.org/browse/ETPLT-7
- https://wiki.jasig.org/display/PLT/EsupSympa

We can thank Christian Murphy and Christian Cousquer in particular for their fine work on this.
While accessibility compliance is a difficult target to hit as portal development continues, we would like to see adoption of ongoing processes that will help us continue to maintain a high degree of accessibility. This should not be viewed only as an attempt to be in compliance with the law; better accessibility generally results in a better experience for all. Just as the move towards a mobile-first design philosophy has resulted in better user interfaces, a move towards accessibile-first implementation will improve the portal for our users. Accessibility is usability.

Improved Internationalization and Localization

Another area of improvement we are working on is better internationalization and localization. Our intention is to have i18n and l10n built into the products; you shouldn't have to fork to internationalize or localize. To accomplish this for all of uPortal is a big task. Our initial focus is on new components. There has also been some research into tools or technologies that could help with this such as CrowdIn. Of course, we are hoping our international colleagues like you will guide and assist us.

Improved Build Architecture

While those are the more visible, user-facing features, there are a number of things we are doing on the backend to make uPortal easier to work with for developers, especially new developers. Our goal is to have uPortal be a collection of smaller projects that are well documented and easy to configure and deploy.

Better Documentation

One way we are looking at improving our documentation is to have it live alongside the code. This will make it a lot easier for developers to create documentation in the first place and then keep it maintained. This process uses Jekyll which generates HTML from Markdown, and that HTML is then hosted on GitHub Pages. As the documentation changes, the site is automatically updated. Since the source of the documentation sits alongside the code in Github, it is also versioned with it. Since the source of the documentation is in Markdown, it's in plain old text files that are easily read and updated, and for which lots of tools have been written. Your favorite text editor almost certainly has excellent Markdown support. Another attractive feature of this plan is that all the technologies used to create the documentation are 100% open source.

At the moment the uPortal documentation is only stubbed out (http://jasig.github.io/uPortal/), but a more complete example of this is available for angularjs-portal (http://uw-madison-doit.github.io/angularjs-portal/).

Common Apereo Binaries

Currently everyone builds their own uPortal executables. In fact, they often rebuild the executable for each environment. We would like to move to genuine shared, versioned, released, collaborated upon software products. You should be able to download a binary from Apereo and easily configure it to work in your deployment. The common binaries should also be created in a way that they fail gracefully if you don't implement a particular function in your environment. While it may take some time to get there, our first steps will include better separation of files so that it is much clearer which files you should leave alone and which files you can look at modifying. By doing a better job at separating these it will be much easier for deployers to internationalize, localize, and configure.

If you absolutely need to locally fix a bug or add an enhancement, you could still build uPortal locally. You would make your fixes on a branch and the submit a pull request to have your fixes incorporated in the next release.

But let's do that as little as possible. If there's a bug that needs to be fixed urgently, let's fix it in the real open source product and cut a release immediately. If there's an enhancement that you really need immediately, let's make it an enhancement available to everyone immediately; implement it in the real actual open source product and cut a release.

Let's get to small shared software products that are semantically versioned and released frequently. Whatever the barriers are to simply enhancing or fixing the real genuine open source product and then adopting that locally as soon as needed, we need to reduce those barriers. This is the path to sustainability, to minimized duplicated effort, and to minimized lost opportunity for collaboration and shared benefit.

Microservices and Container Packaging

And finally, we are moving toward uPortal being a collection of smaller projects and microservices. These smaller projects are easier for developers to understand and thus easier to create and maintain. It would be easier for a new developer to start contributing by focusing on a smaller piece without having to understand the entire uPortal system. This also makes it easier or more flexible for deployers. By breaking things down into smaller services, it is possible for a university to only deploy the services they need. We would also like to see some of these components being packaged in a way that they can easily be deployed in containers like Docker.

Smaller pieces mean less friction in fixing a bug or adding a feature, and releasing. Smaller pieces mean less friction to getting internationalization and localization right, at least in that small piece. Smaller pieces mean a richer, more sustainable, more participatory ecosystem. You, as a developer, or institution, don't have to commit to understanding and progressing the entire giant open source portal stack to contribute effectively. Pick just one small piece to adopt, to shepherd, to grow. And this doesn't just go for software developers; this goes for everyone. You can pick a small piece you like, you care for, and add value. Document. Test. Identify opportunities for improvement, use cases, a roadmap.

We are more than the sum of our parts by dividing and conquering the parts.

Working Together

And that brings me to my final topic. I would like to spend the remaining time discussing how we can work together more closely. I'll start with two general recommendations and then I want to hear from you.

Share Early and Often

First and foremost, I encourage you to share your ideas, feedback, and questions in public. We are a very welcoming community. The easiest place to share something is by posting on the discussion lists. It could be simply an idea you have on how to improve part of uPortal or it could be something that you are working on for your university that you think others may be interested in. It is a way that you can get feedback or potentially find collaborators.

We have also seen examples of people blogging about topics and then posting a link to the entry on the discussion list. We may also see more use of the Apereo GitHub blog, https://apereo.github.io/. While the scope and purpose of that blog are still in flux, there is nothing stopping you from submitting a new entry as a Pull Request.

We would also like to see more hangouts, webinars, and community calls. Sometimes these may be structured but sometimes they can just be people gathering online to discuss a topic. Anyone is free to create an online event and invite others to participate.

Branch, Don't Fork

My second recommendation is to branch and not fork. At first it seems easier to fork the project and add your own customizations. We have found, however, that in the long run it pays off to branch as necessary and try to move your enhancements into the project itself. The more we work from a single code base, the easier it is. We shouldn't have separate versions of uPortal, "The Wisconsin Version" or "The ESUP Version" - we should simply have different deployments of the same code. This will really help us move forward together faster.

Definitely don't fork, but consider not branching either. Not really. Not for more than a moment.

There's a technical way of using branches to compose a proposed changeset and offer it as a Pull Request, as a patch to be applied to the project. Yes, of course do that. But then let's accept that Pull Request, let's merge that change, and let's cut a release, and then adopt that release. Don't locally fork. Don't even run a local

branch. Run real releases as they come, and if they're not good enough to run as they come, then let's make them good enough to run as they come.

Discussion

But now I would like to hear more from you on how we can work better together.
What are the barriers to more adoption and collaboration?
What ideas do you have that would make it easier?
What are your ideas for the future of uPortal?

Resources

Presentation slides: https://goo.gl/U9R99O
Presentation notes: https://goo.gl/D5BkW2
uPortal 5 overhaul: https://wiki.jasig.org/x/GYBfB
Beyond Portals presentation from Educause: https://goo.gl/6rtVd7